Concepts Objets

Université Hassan 1er
Faculté des Sciences et Techniques

Département de Mathématiques et Informatique

Principaux modèle de représentation du monde :

- Programmation fonctionnel ou Procédurale
- Programmation par GoTo (aller à)
- Approche Orientée Objet (que nous allons étudier).

Critères de qualité d'un logiciel (Programme)

1. Correct
2. Robuste
3. Extensible
4. Réutilisable
5. Portable
6. Efficace

Programmation par GoTo

- Saut inconditionnel
- Lisibilité, maintenance et réutilisation
- L'instruction goto est une instruction présente dans de nombreux langages de programmation. Elle est utilisée pour réaliser des sauts inconditionnels dans un programme. L'exécution est renvoyée vers une étiquette ou label, qui est soit un numéro de ligne donné, soit une étiquette déclarée, selon le langage.

Exemple :

Calcul de N! avec l'instruction goto (langage C)

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int fact, N, i;

    label_saisie :  printf("Donnez N >=0\n");
                    scanf("%d",&N);
    if(N<0)         goto label_saisie;

    fact=1; i=1;
    label_calcul :  fact=fact*i;
                    i++;
    if(i<=N)        goto label_calcul;
    printf("%d!= %d\n", N, fact);
    system("PAUSE");
    return 0;
}
```

Le modèle fonctionnel: Principe/Avantages/Limites

1. Découpage du problème en fonctions
2. Les tâches avant les données
3. Spécialisation des fonctions
4. Vérification des critères de qualité

Dans une approche fonctionnelle, vos programmes sont composés d'une série de fonctions, qui assurent ensemble certains services.
Il s'agit d'une approche logique, cohérente et intuitive de la programmation. (C, Matlab, …)

Cette approche a un avantage certain appelé la factorisation des comportements (c'est-à-dire pour créer une fonction d'une application, rien ne vous empêche d'utiliser un autre ensemble de fonctions (qui sont donc déjà écrites)).

Exemple:

```c
#include <stdio.h>
#include <stdlib.h>
#include "dec3.h"
int main(int argc, char *argv[])
{ int val;
  val=LectureEntier();
  printf("valeur lu = %d",val);
  system("PAUSE");
  return 0;
}
```

```c
int LectureEntier(void)
{ int a;
  printf("Donnez un entier\n");
  scanf("%d",&a);
  return(a);
}
```

main.c

dec3.h

Mais , l'approche fonctionnelle a aussi ses défauts, comme par exemple une maintenance complexe en cas d'évolution de votre application (une simple mise à jour de l'application à un point donné peut impacter en cascade sur d'autres fonctions de notre application).

L'application sera alors retouchée dans sa globalité.

L'approche fonctionnelle n'est pas adaptée au développement d'applications qui évoluent sans cesse et dont la complexité croit continuellement (plusieurs dizaines de milliers de lignes de code).

Dans un langage comme Pascal ou C, un programme est une suite d'instructions qui peuvent être organisées en des parties de taille convenable, appelées procédures ou fonctions, qui accomplissent des tâches précises.

Lors de l'exécution, ces procédures utilisent des données. Il y a dissociation entre données et procédures, ce qui conduit à des problèmes lorsqu'on change les structures de données.

Les procédures s'appellent entre elles et peuvent modifier les mêmes données. Ce qui peut causer des problèmes lorsqu'on veut modifier une procédure : Comment avait elle été appelée ?

Approche Orientée Objet

Le concept de programmation orientée objet vient donc pour répondre à un certain nombre d'objectifs dont :

- Similarité avec le monde réel (Approche Intuitive)
- Similarité avec le langage naturel
- Utilisable tout au long du processus méthodologique
- Abstractions
- Réduction de la complexité, modularité, autonomie, contrat
- Réutilisation, flexibilité, robustesse
- Prise en compte des systèmes existants
- Prise en compte de l'évolution du besoin

Évolution des langages de programmation

Les programmes deviennent de taille de plus en plus grande et nécessitent de plus en plus la collaboration d'un grand nombre de développeurs.

L'évolution rapide des besoins et technologies fait que les programmes doivent être faciles à tester, à améliorer, à réutiliser et à maintenir.

Le concept de programmation orienté objet vient de la volonté d'avoir une représentation très proche du monde réel. En effet, ce monde est constitué d'objets. Ils sont caractérisés par leur dimension, leur poids, leur couleur, etc.

Évolution des langages de programmation

Une molécule, une voiture, un compte bancaire ou un étudiant sont autant d'objets que l'on rencontre en permanence.

Ce lien étroit entre approche objet et monde réel fait qu'informaticien et non informaticien peuvent avoir un langage commun basé sur ce concept.

Pour résumer, un objet permet de désigner une représentation "abstraite" d'une chose concrète du monde réel ou virtuel.

Historique

- Fonctions et procédures (60 Fortran)
- Typage des données (70) Pascal Algol
- Modules: données + fonctions regroupées

Programmation objet: classes, objets et héritage …

- Simula a été le premier langage de programmation à implémenter le concept de classes en 1967 !

- En 1976, Smalltalk implémente les concepts d'encapsulation, d'agrégation, et d'héritage (les principaux concepts de l'approche objet).

- D'autre part, de nombreux langages orientés objets ont été mis au point (Object Pascal, C++, Java, etc.).

Le modèle Objet : Principe/Avantages

Un objet est une abstraction d'un élément du monde réel. (Voiture→ objet) possède des informations, par exemple Couleur, Matricule, modèle, etc., et se comporte suivant un ensemble d'opérations qui lui sont applicables: vitesse, accélérer, freiner, tourner, etc.

De plus, un ensemble d'attributs caractérisent l'état d'un objet, et l'on dispose d'un ensemble d'opérations (les méthodes) qui permettent d'agir sur le comportement de notre objet.

Les avantages de l'approche objet

- La modélisation des objets de l'application : informatiquement un ensemble d'éléments d'une partie du monde réel en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets
- La modularité (la programmation modulaire permet la réutilisation du code, via l'écriture de librairies)
- La réutilisation
- L'extensibilité
- (pour une meilleure productivité des développeurs et une plus grande qualité des applications).

- Programmation orientée objet apporte l'indépendance entre les programmes, les données et les procédures parce que les programmes peuvent partager les mêmes objets sans avoir à se connaître comme avec le mécanisme d'import/export.

- L'approche objet : a introduit indépendamment de tout langage de programmation à l'objet trois concepts de base : objet, classe et héritage entre classes. Les concepts objet et classe sont interdépendants, c'est-à-dire qu'un objet est une instance d'une classe et la classe décrit la structure et le comportement communs d'objets (ses instances).

Qu'est-ce qu'un objet ?

En programmation orientée objet, chaque entité du monde réel doit être représentée justement par un objet.

A chaque instant, cette entité (objet) est caractérisée par :

- L'information qui ressort de la valeur des données (son état : attribut),
- Les opérations qui peuvent modifier son état ou interagir avec les autres objets (méthodes),
- L'identité qui caractérise l'existence propre d'un objet et permet de distinguer deux objets dont toutes les valeurs d'attributs sont identiques.

OBJET = Etat + Comportement + Identité

OBJET = Attributs+Méthodes

Objet : C'est une structure comportant

- des données (l'état de l'objet)
- des services (le comportement de l'objet)

Un **Objet** peut correspondre à :

- **Objet concret** du monde réel, ayant une réalité physique;
- Un **concept abstrait**;

Bref

- objet = instance de classe
- = description d'un élément particulier
- manipulé par des référence
- cycle de vie d'un objet

La notion de classe

- Une **Classe** est un modèle informatique représentant une famille d'objets ayant :

✓ La même structure de données (même liste d'attributs)

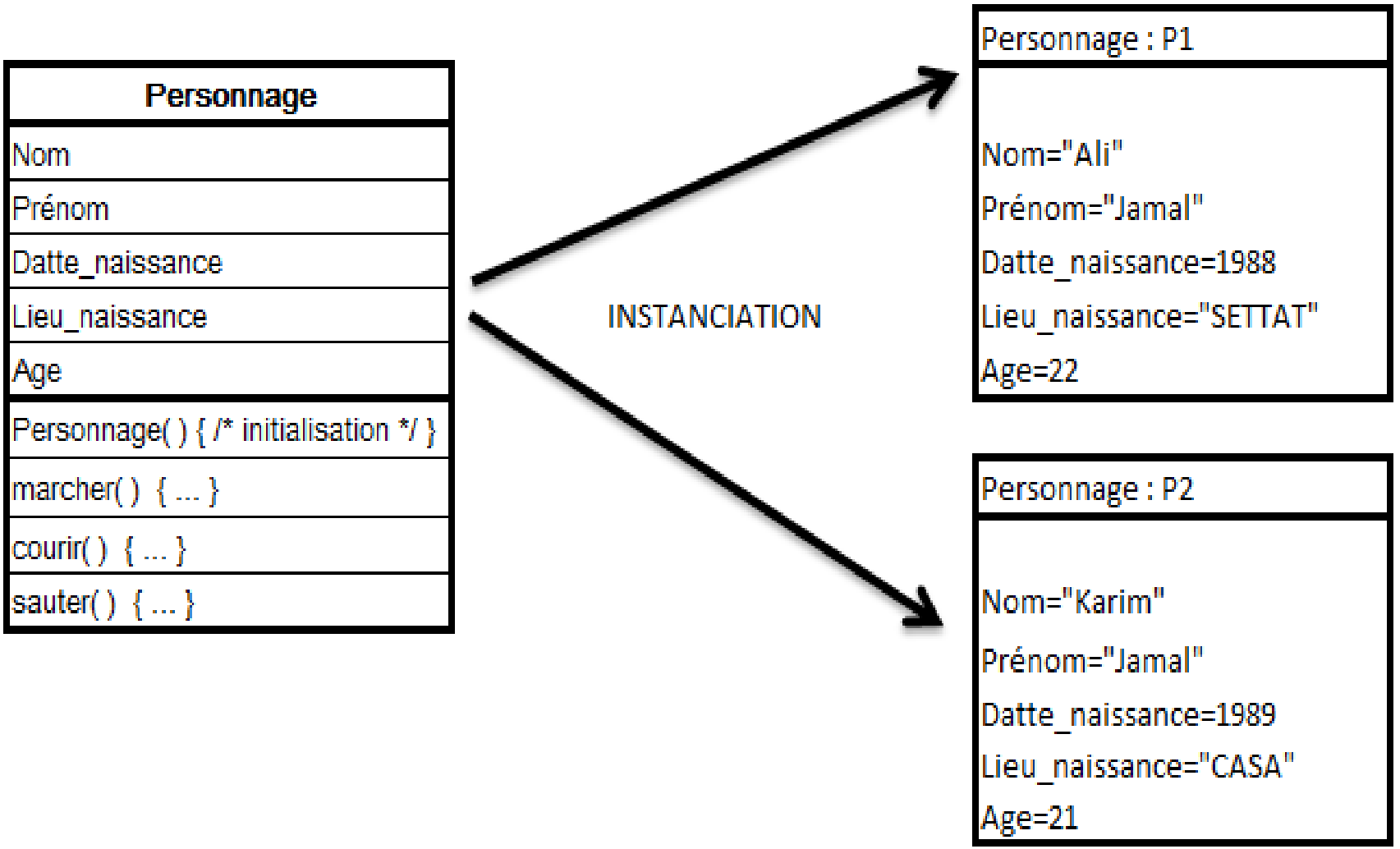
✓ Les mêmes méthodes (mêmes comportements).

- La Classe par elle-même ne contient pas les valeurs des données : c'est un type de données abstrait.

- La notion de **Classe** est fortement liée à la notion de **type.**

- La création d'un objet en tant qu'exemplaire concret (contenant des données) d'une classe s'appelle une **INSTANCIATION**.
- Chaque objet donne des valeurs aux attributs de la classe.

Exemple


Personnage
Nom
Prénom
Datte_naissance
Lieu_naissance
Age
Personnage( ) { /* initialisation */ }
marcher( ) { ... }
courir( ) { ... }
sauter( ) { ... }
INSTANCIATION
Personnage : P1
Nom="Ali"
Prénom="Jamal"
Datte_naissance=1988
Lieu_naissance="SETTAT"
Age=22
Personnage : P2
Nom="Karim"
Prénom="Jamal"
Datte_naissance=1989
Lieu_naissance="CASA"
Age=21

- L’instanciation : L’objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. L'instanciation représente la relation entre un objet et sa classe d'appartenance qui a permis de le créer.

- Les attributs (appelés aussi variables d'instances): Ils ont un nom et soit un type de base (simple ou construit) soit une classe (l'attribut référence un objet de la même ou une autre classe).

- Les opérations (appelées parfois méthodes): Elles sont les opérations applicables à un objet de la classe. Elles peuvent modifier tout ou en partie l'état d'un objet et retourner des valeurs calculées à partir de cet état.

Attribut

Définition : propriété définie par un nom, un type et éventuellement une valeur initiale.

- visibilité
- nom attribut : identificateur de l'attribut, unique au sein de la classe.
- multiplicité : l'attribut représente un ensemble de valeurs
- exemple de tableau: Parents[1..2] : Personne.
- valeur initiale : valeur prise par l'attribut lors de l'instanciation de la classe (valeur en concordance avec le type de l'attribut).

Attribut : Type

Type d'attribut : ensemble des valeurs pouvant être prises par l'attribut. Il peut être :

- Une classe : Rectangle, Cercle, TV, Voiture, Personne, …
- Un type primitif : Entier, Chaîne de caractères, Booléen, …
- Une expression complexe

Opération

Définition : spécification du comportement (objets) des instances de la classe.

Cinq catégories d'opérations :

- Constructeurs qui créent les objets,
- Destructeurs qui détruisent les objets,
- Sélecteurs (opérations de consultation) qui renvoient tout ou une partie de l'état d'un objet,
- Modificateurs qui changent tout ou partie de l'état d'un objet,
- Itérateurs qui visitent l'état d'un objet ou le contenu d'une structure de données contenant des objets.

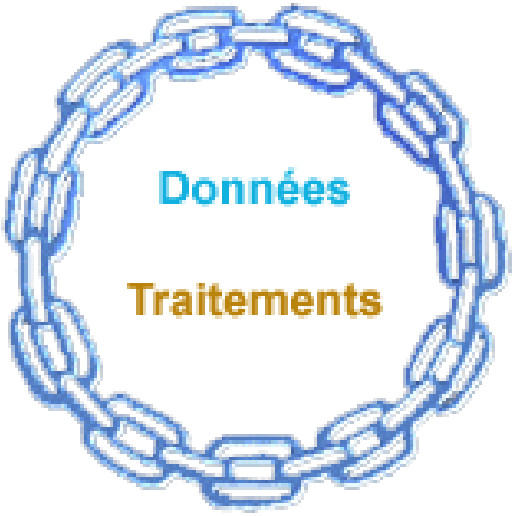
Interface

Ce que l'on vient de faire = description d'une interface

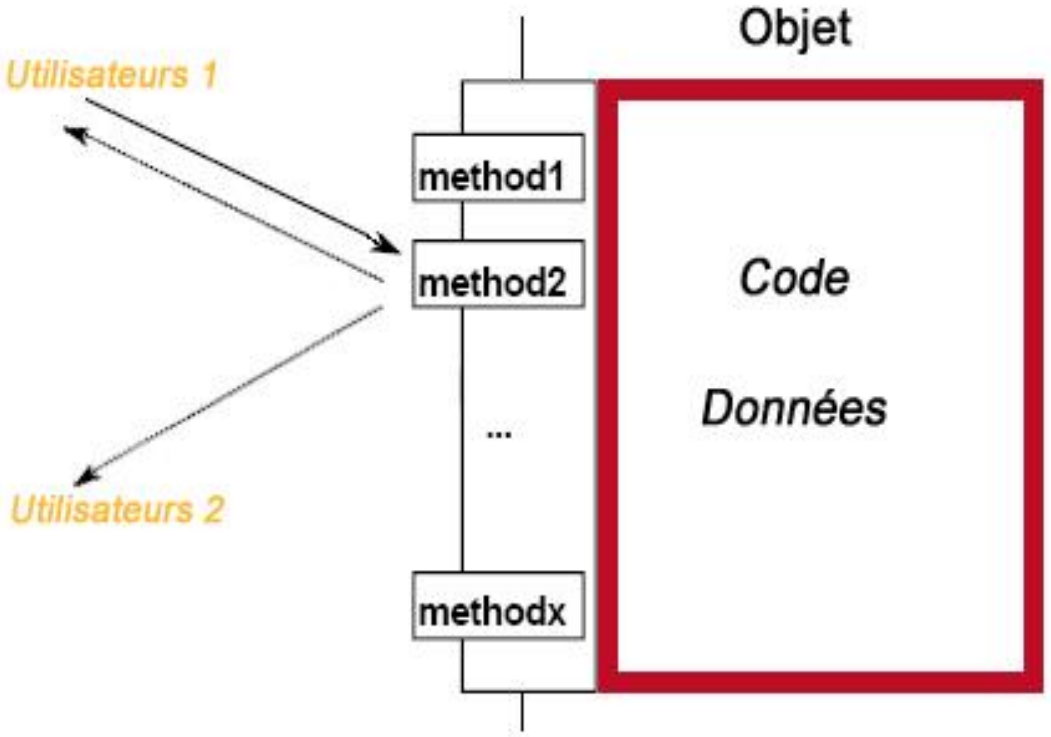
- Ce que fait un objet
- Comment il le fait (implantation)
- Interface = contrat = spécification

Intérêt

- On a juste besoin de connaître les interfaces et pas toute l'implantation
- Programmation de plus haut niveau (meilleure lisibilité)



Objet : interface (partie publique) + données (partie privée)



L'utilisateur accède à l'objet à travers son interface

Le concept d'encapsulation

- L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure (classe) en cachant l'implémentation de l'objet

- L'encapsulation empêche également les utilisateurs clients de connaître les détails de l'implémentation en ne fournissant qu'une vue externe d'un objet. Seule l'interface d'un objet doit être visible à un client potentiel.

- Dans l'approche Objet : La modularité est prise en compte par l'encapsulation. L'unité de modularité est la classe. Les classes peuvent être regroupées en **packages** ou en **sous systèmes**

Le concept d'encapsulation

Codeur:

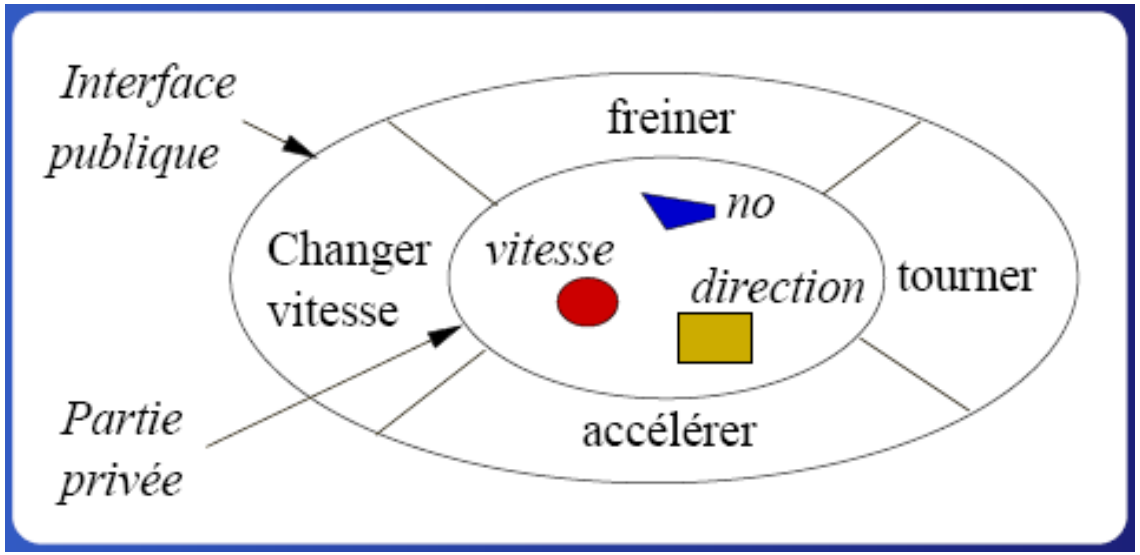
- la séparation entre interface et implantation permet de s'abstraire des détails de fonctionnement
- protection de l'information, …

Le client :

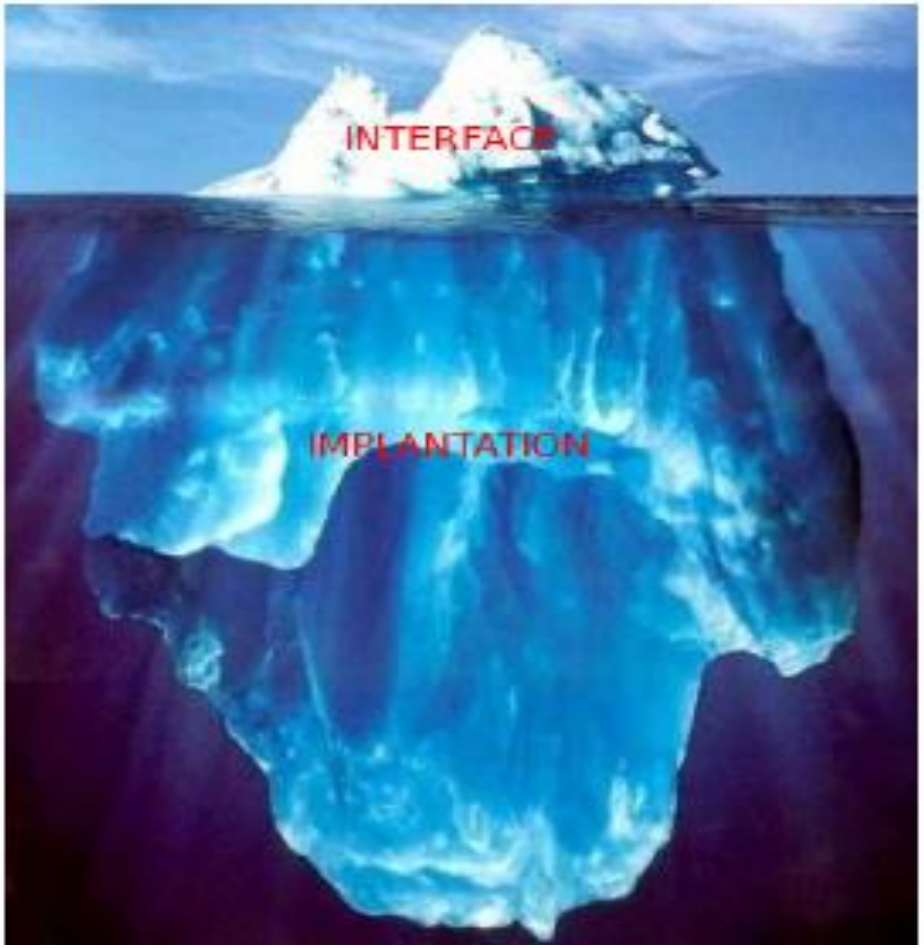
- ne manipule que l'interface
- n'a pas accès aux données /manière dont sont stockées les éléments

Le concept d'encapsulation

- C'est la capacité à *utiliser* un objet sans savoir comment il est implanté.



Le concept d'encapsulation



Avantages de l'encapsulation

- **Évolutivité** : on peut rendre en cause l'implantation de manière transparente.
- **Modularité** : le code est écrit et maintenu indépendamment du reste de l'application.
- **Simplicité** : toute la complexité doit être masquée
- **Sécurité** :
  - → Les données sont accessibles seulement via les services
  - → Ne montrer que l'essentiel : l'interface

Exemple

Un compte bancaire peut être décrit par le nom, prénom, adresse du client, le numéro de compte et un solde et le découvert autorisé.

Les clients peuvent effectuer des retraits ou des dépôts d'argent, changer d'adresse ou fermer leur compte. La banque peut calculer des intérêts et appliquer des frais de transactions.

Exemple

Classe Compte_bancaire

int numéro

double solde

depot ( )

retrait ( )

estADecouvert ()

```
class CompteBancaire
{   int      numero; // Numéro du compte bancaire
    double solde ;    // Solde du compte bancaire
    void depot (double montant) {solde = solde + montant ;}
    void retrait (double montant) {solde = solde - montant ;}
    boolean estADecouvert () {return solde < 0 ;}
}
```

```
public class Banque {
  public static void main (String arg[]) {
   CompteBancaire cptDurand ;
   cptDurand = new CompteBancaire( );
   cptDurand.numero = 34687;
   System.out.println ("Solde initial : " + cptDurand.solde);
   cptDurand.depot (1500.00);
   System.out.println ("Solde apres depot : " + cptDurand.solde);
   cptDurand.retrait (2000.00);
   System.out.println ("Solde apres retrait : " + cptDurand.solde);
   if (cptDurand.estADecouvert())
      System.out.println ("Attention au compte " + cptDurand.numero);
    }
}
```

Visibilité

Il existe trois niveaux de visibilité :

- publique : Les fonctions de toutes les classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité public. Il s'agit du plus bas niveau de protection des données.
- protégée : l'accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées.
- privée : l'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé.

Exemple d'une classe en java

- Classe représentant un point de R^2
- Opération : translater le point d'un vecteur (tx, ty)

```
public class Point
{
        private double x;
        private double y;
        public void translate(double x, double y)
       {
         this.x += x;
         this.y += y;
        }
}
```

1- public class Point

- définition d'une nouvelle classe Point (dans un fichier Point.java)
- modèle général de n'importe quel point de R^2

2- private double x;

- attributs de la classe
- = données / variables
- = décrivent un état
- mode d'accès private = accessible uniquement depuis cette classe
- accès **à l'intérieur de la classe** : this.nomAttribut

3- public void translate(double tx, double ty)

- méthode de la classe
- même principe qu'une fonction en C : type de retour + paramètres typés
- mode d'accès public = accessible à l'extérieur de la classe
- Remarque : on a fait le choix de représenter un point par ses coordonnées cartésiennes ;

Cycle de vie des objets

- création

  création d'une nouvelle instance lorsque l'on appelle new

- utilisation :

  nomReference.nomMethode()

- destruction

  automatique dès qu'il n'y a plus de référence qui « pointe » sur l'objet en question

Constructeur

- constructeur = méthode particulière permettant de créer des instances de la classe
- objectif = initialiser les différents attributs
- définition d'un constructeur =
  - même nom que la classe
  - pas de type de retour
  - autant de paramètres que ce dont on a besoin

Destructeur virtuel

- Un destructeur peut être défini comme virtuel. Cependant, un constructeur ne pourra jamais l'être.

- Lorsque l'on détruit un objet de la classe dérivée qui est référencé par un pointeur de la classe de base, alors le destructeur de la classe dérivée est appelé .

- Sinon seul le destructeur de la classe de base est appelé.

Destructeur virtuel

```
ObjetGeo *OB = new Cube();

delete OB;
```

Si le destructeur de ObjetGeo est virtuel, le destructeur de Cube pourra être appelé en premier. Sinon, seul le destructeur de ObjetGeo sera appelé.

Communication entre objets

- Objets pouvant échanger des **messages** avec d'autres objets
- Le comportement global d'un système repose sur :

  - l'ensemble des objets identifiés ;

  - les relations et les communications possibles entre ces objets.
- Les objets **interagissent entre eux** pour réaliser les **fonctionnalités** de l'application.
- Un MESSAGE:

  - transporte l'information nécessaire aux demandes à satisfaire,

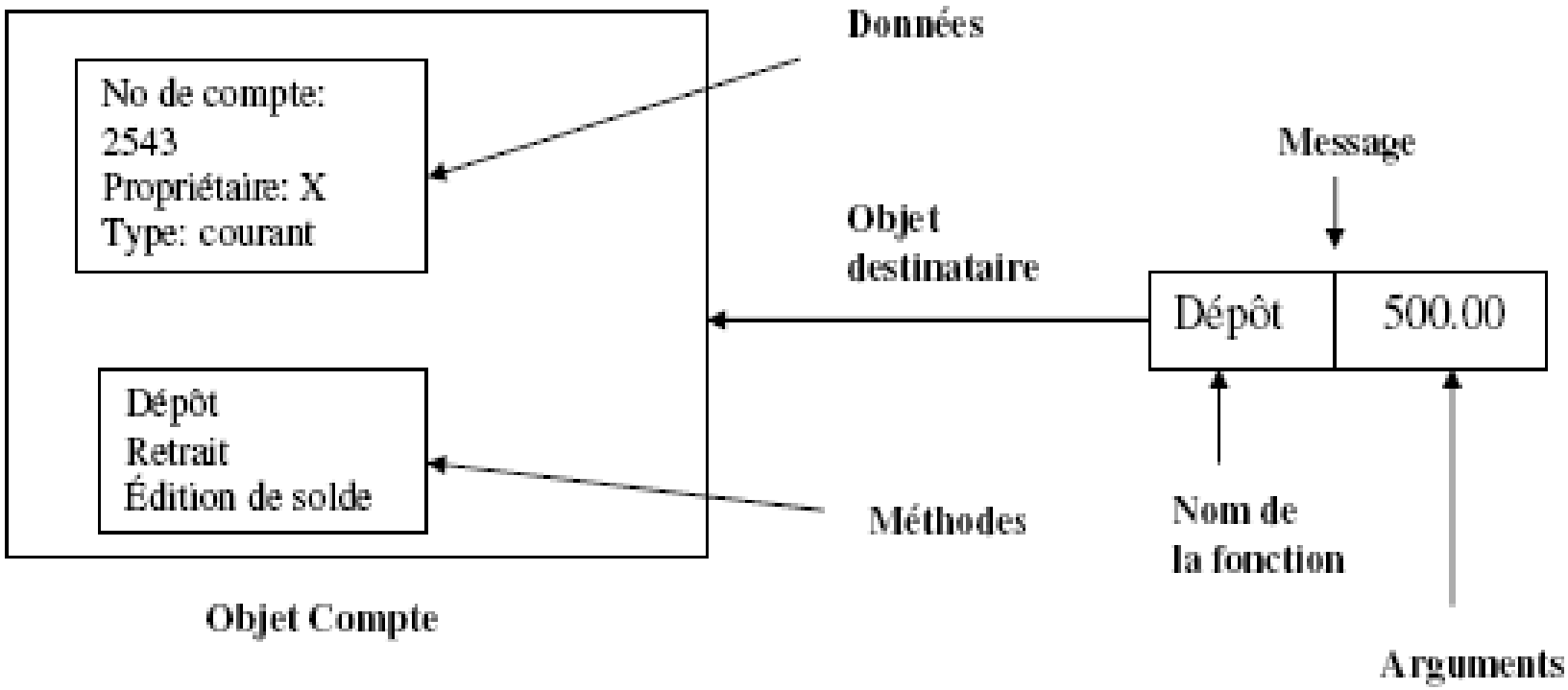
  - moyen UNIQUE de communication avec les objets (impossible d'accéder directement aux données encapsulées d'un objet).
- Contenue:

  - Nom de l'objet destinataire,

  - Enoncé de la demande (exemple: le nom d'une fonction),

  - Les arguments nécessaires (pour réaliser la demande)

Communication entre objets



- Envoyer un message à un objet, c'est lui demander d'exécuter une de ses méthodes

Surcharge de méthodes

- L'identifiant d'une méthode est défini par sa signature :
  - nom choisi par le programmeur
  - liste des types des arguments
  - ne prend pas en compte le type retour
- Deux méthodes différentes peuvent avoir le même nom si elles se distinguent par leurs signatures (surcharge)
- L'interpréteur choisit la bonne méthode en fonction :
  - Du nombre d'arguments
  - Du type des paramètres figurant dans l'invocation
  - Le type de retour n'est pas pris en compte

Surcharge de méthodes

- Il est interdit en C de définir plusieurs fonctions qui portent le même nom. En C++, cette interdiction est levée, moyennant quelques précautions. Le compilateur peut différencier deux fonctions en regardant le type des paramètres qu'elle reçoit.

- La liste de ces types s'appelle la signature de la fonction.

- Le type du résultat de la fonction ne permet pas de l'identifier, car le résultat peut ne pas être utilisé ou peut être converti en une valeur d'un autre type avant d'être utilisé après l'appel de cette fonction.

Intérêt de la surcharge

- Permet d'avoir des méthodes qui font des choses similaires sur des données de type différent

Exemple:

add(float f, float g) et add(int f, int g)

plus clairs / compacts que: add_float et add_int

Fonctions inline

- Le C++ dispose du mot clef **inline**, qui permet de modifier la méthode d'implémentation des fonctions. Placé devant la déclaration d'une fonction, il propose au compilateur de ne pas instancier cette fonction. Cela signifie que l'on désire que le compilateur remplace l'appel de la fonction par le code correspondant.
- De plus, il faut connaître les restrictions des fonctions inline :

- Elles ne peuvent pas être récursives ;

- Elles ne sont pas instanciées, donc on ne peut pas faire de pointeur sur une fonction inline.

Fonctions inline

Exemple :

```
 inline int Max(int i, int j)
{
    return (i>j  ?  i  :  j);
}
```

Abstraction

- La notion d'abstraction consiste à identifier pour un ensemble d'éléments :

  - Des caractéristiques valides pour tout les éléments
  - Des mécanismes communs à tout les éléments

- Le choix des attributs et méthodes dépend du niveau de modélisation que l'on souhaite

Relations entre classes

- Les relations entre classes peuvent être de plusieurs natures :

  → L'Héritage, traduit une classification ;

  → La délégation est une relation de type "a un" entre deux objets. Elle intervient lorsque deux objets collaborent entre eux. Ce type de relation se décompose en trois parties :

  1- L'association

  2- L'agrégation

  3- La composition

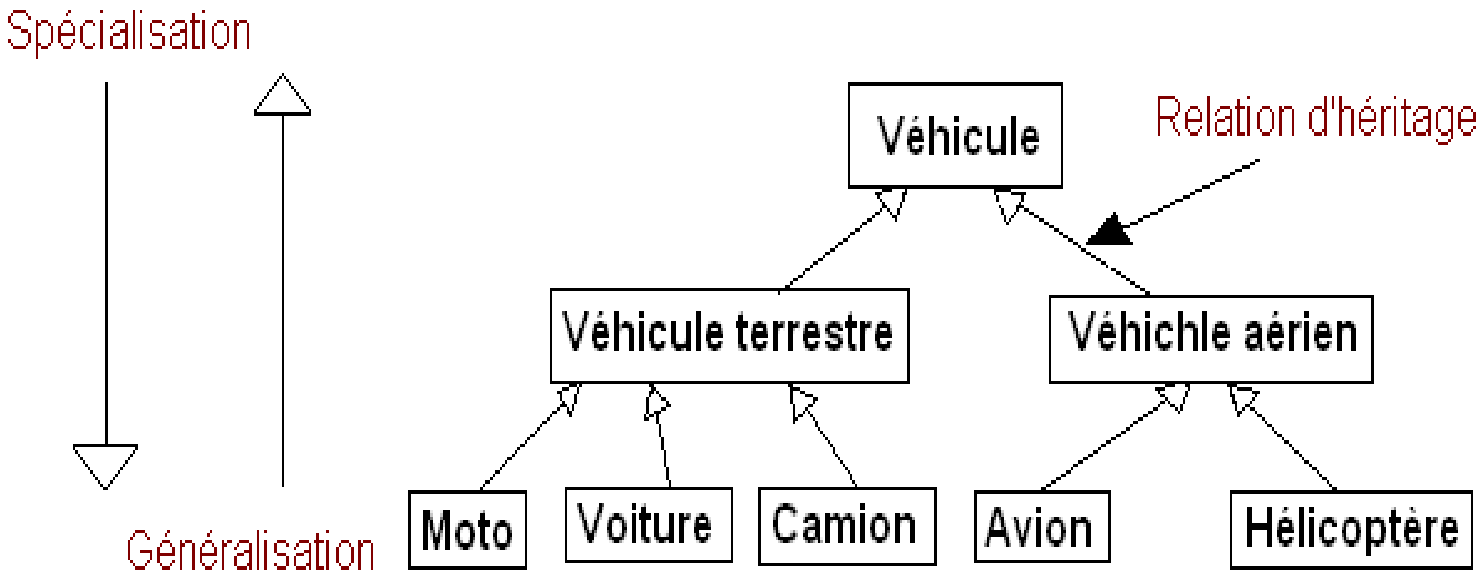
- Les relations entre classes modélisent les interactions (les "liens") entre objets

Héritage

- L'héritage (en anglais inheritance) est un principe propre à la programmation orientée objet, permettant de créer une nouvelle classe à partir d'une classe existante. Le nom d'"héritage" (ou parfois dérivation de classe) provient du fait que la classe dérivée (la classe récemment créée) contient les attributs et les méthodes de sa superclasse (la classe dont elle dérive).
- L'intérêt majeur de l'héritage est de pouvoir définir de nouveaux attributs et de nouvelles méthodes pour la classe dérivée, qui viennent s'ajouter à ceux et celles héritées. Par ce moyen on crée une hiérarchie de classes de plus en plus spécialisées.
- Héritage mécanisme de transmission des propriétés d'une classe vers une sous-classe.

Héritage

- Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.



Héritage

- **Classe Véhicule terrestre**

  Immatriculation

  Marque

  Puissance

  Rouler()

  Démarrer

  Stopper()

- Classe Voiture

  Immatriculation

  Marque

  Puissance

  Qté d'essence

  Nombre de places

  Nombre de portes

  Rouler()

  Démarrer

  Stopper()

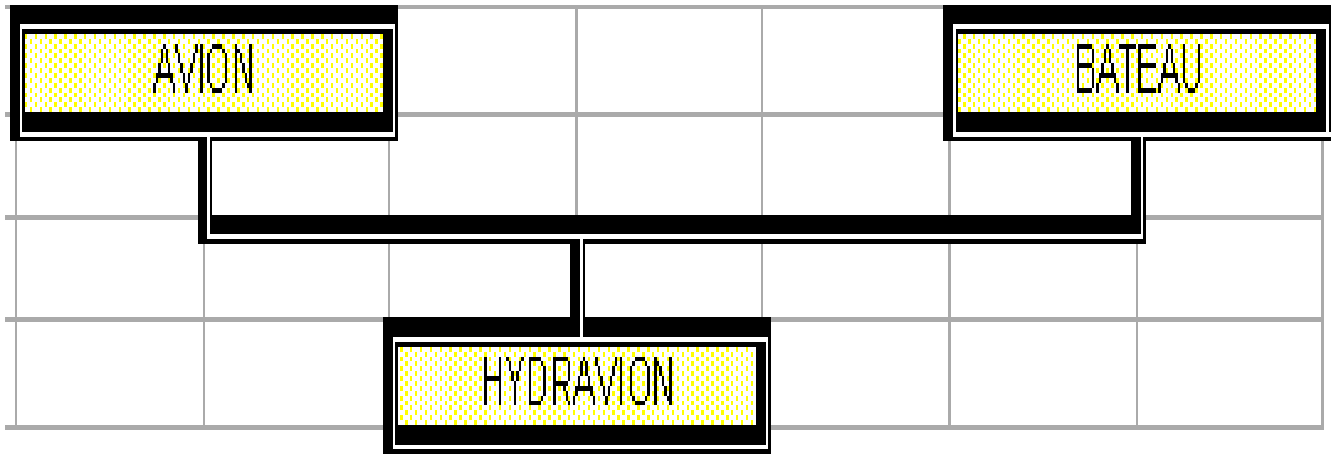
Héritage

- Certains langages orientés objet, tels que le C++, permettent de faire de l'héritage multiple, ce qui signifie qu'ils offrent la possibilité de faire hériter une classe de deux superclasse. Ainsi, cette technique permet de regrouper au sein d'une seule et même classe les attributs et les méthodes de plusieurs classes.

- L'héritage multiple représente le mécanisme par lequel une sous-classe hérite des propriétés de plus d'une super-classe.

- Une classe fille hérite de plusieurs classes mères. Elle hérite d'une partie des attributs et des méthodes de chacune de ses classes mères, en plus de ses spécifications propres.

Héritage



Polymorphisme

- Le nom de *polymorphisme* vient du grec et signifie *qui peut prendre plusieurs formes*. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

- On distingue généralement trois types de polymorphisme :

  Le polymorphisme ad hoc (*surcharge*)

  Le polymorphisme paramétrique (*généricité*)

  Le polymorphisme d'héritage (*redéfinition*, *spécialisation*)

Le polymorphisme ad hoc

- Le polymorphisme ad hoc permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles (si ce n'est bien sûr d'être des filles de la classe objet).
- Par exemple : Faculté qu'on des objets différents de réagir différemment en réponse au même message.

- Marcher sur la queue d'un chat => il miaule,

- Marcher sur la queue d'un chien => il aboie,

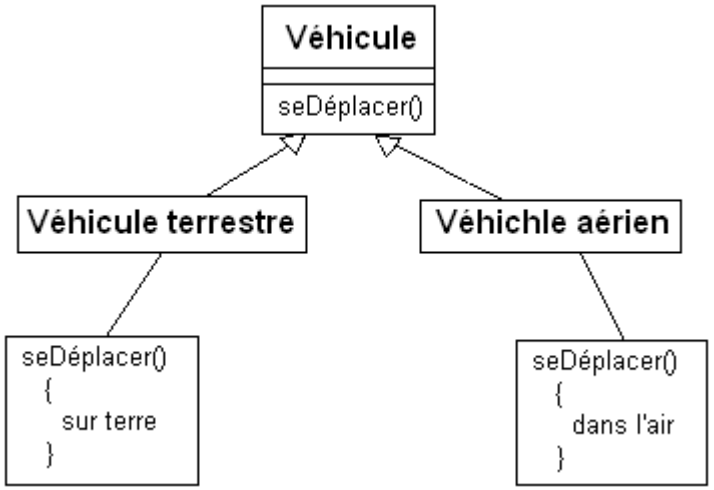
Le polymorphisme paramétrique

- Le polymorphisme paramétrique, appelé généricité, représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type). Le polymorphisme *paramétrique* rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.
- Ainsi, on peut par exemple définir plusieurs méthodes homonymes *addition()* effectuant une somme de valeurs.

Fonction: opération addition (+)
L'addition des nombres entiers :
1 + 2 = 3
L'addition des nombres complexes :
1+2i + 3+4i = 4+6i

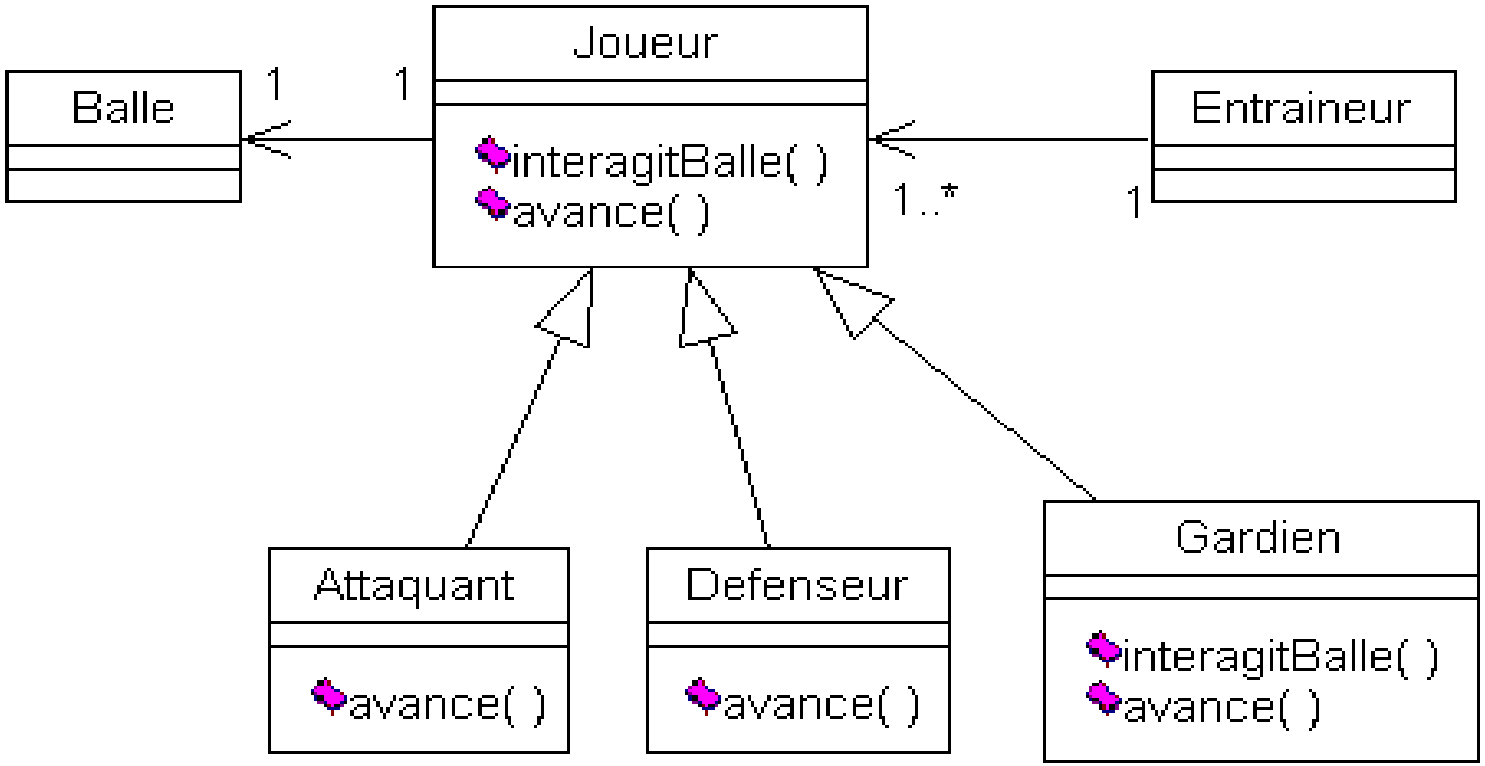
Le polymorphisme d'héritage

- La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la **spécialisation**. Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque : il s'agit du **polymorphisme d'héritage**. Ceci permet de faire abstraction des détails des classes spécialisées d'une famille d'objet, en les masquant par une interface commune (qui est la classe de base).



Le polymorphisme d'héritage

- Un match de foot polymorphique



Le polymorphisme d'héritage

- Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.
- La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.
- L'héritage évite la duplication et encourage la réutilisation.
- Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
- Le polymorphisme augmente la généricité du code.

Délégation

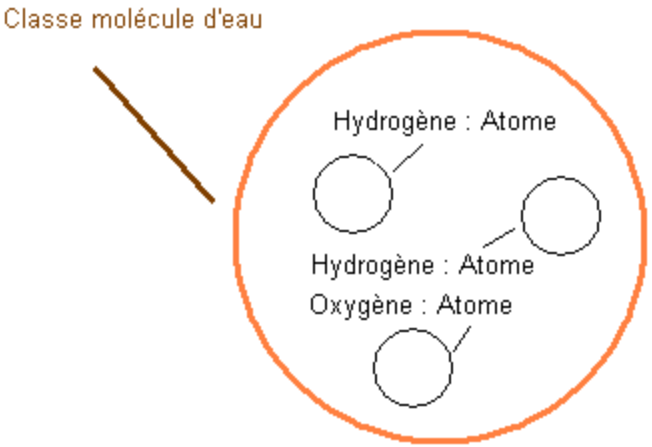
- La délégation décrit un comportement particulier entre objets. Il s'agit d'une délégation au "sens commun" du terme. Un objet, qui a la responsabilité d'accomplir une tâche, et qui se voit demander d'accomplir une tâche, peut disposer d'une méthode qui lui permet de déléguer cette tâche à un autre objet, qui l'effectuera a sa place.
- Ce type de relation se décompose en trois parties :

  1- L'association : elle consiste à une simple relation de collaboration entre deux objets. Chaque objet existe de manière totalement indépendante, et n'a pas forcément besoin de l'autre pour "vivre". La relation entre un téléphone portable et son kit main-libre est une relation d'association.

Délégation

2- L'agrégation : elle se distingue de l'association par le fait qu'un objet est une partie de l'autre. Ainsi, un bouton sur un téléphone portable a une relation d'agrégation avec le téléphone. Si une certaine touche n'est pas présente, le téléphone peut néanmoins fonctionner, en cachant ce manque par des combinaisons d'autres touches.

L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes



3- La composition : sous-partie de la relation d'agrégation. Elle représente une relation étroite entre les deux objets : l'objet "père" ne peut fonctionner sans l'objet "fils". L'objet "batterie" ou "micro" d'un téléphone portable ont une relation de composition avec le téléphone, car le téléphone ne peut pas fonctionner sans "batterie" ou "micro" .

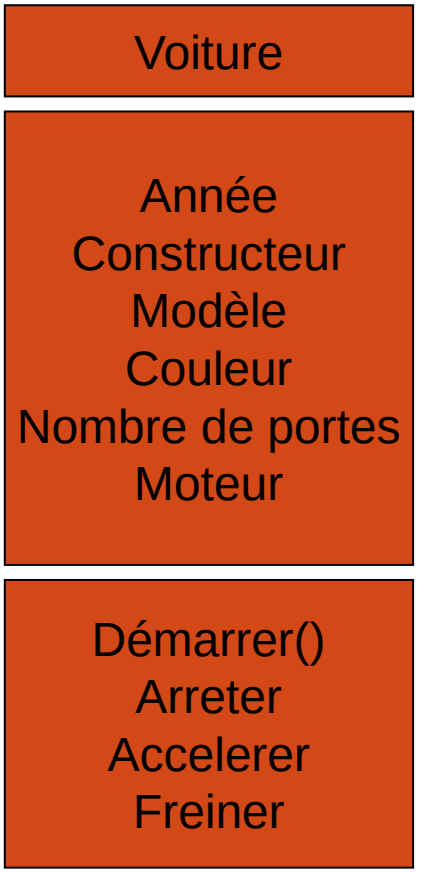
Résumé

Classe

Une Classe définit les caractéristiques d'un Objet.

Les caractéristiques :

- Attributs (champs, propriétés, données)
- Comportement (méthodes ou opérations)



Objet

- Un Objet est une instance d'une classe
- Un Objet est la concrétisation d'une classe
- La création d'un objet à partir d'une Classe s'appelle : INSTANTIATION

maPorsche : Voiture

Année = 2007
Constructeur = Porsche
Modèle = Carrera GT
Couleur = Gris
Nombre de portes = 2
Moteur = 5.7 L V10

Méthode

- Une méthode définit un comportement d'un Objet
- Une méthode affecte en général un seul Objet

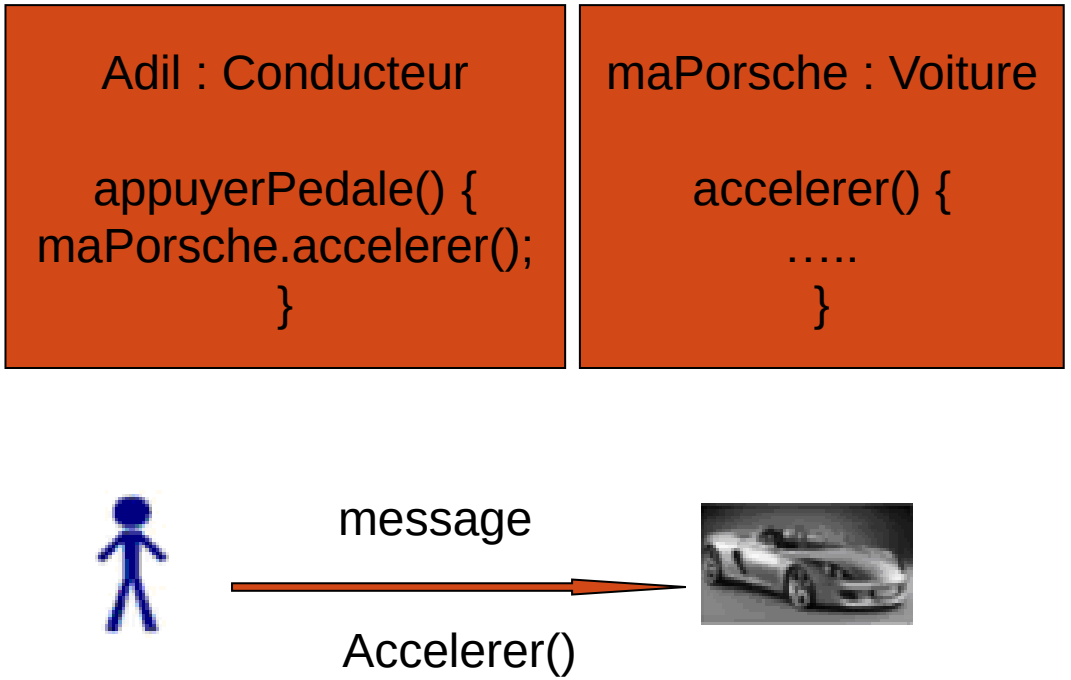
maPorsche : Voiture

Année = 2007
Constructeur = Porsche
Modèle = Carrera GT
Couleur = Gris
Nombre de portes = 2
Moteur = 5.7 L V10

Démarrer()
Arreter()
Accelerer()
Freiner()

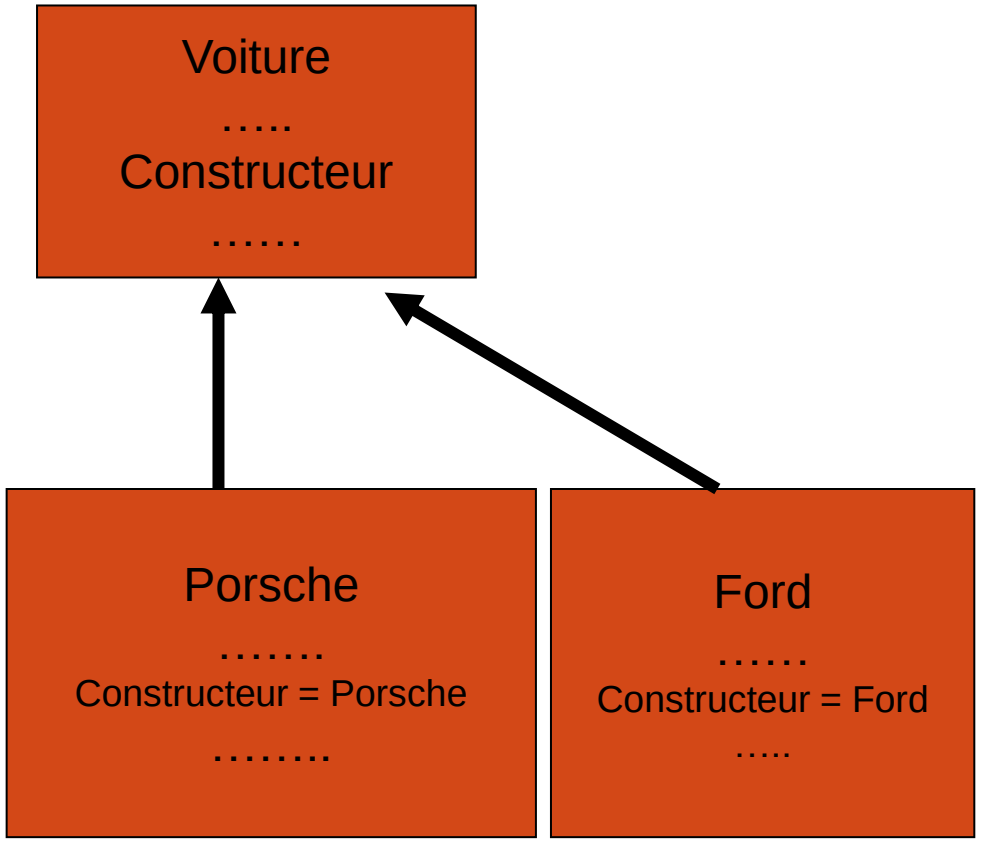
Envoi de message

- Processus où un Objet envoie une donnée à un autre Objet ou lui demande d'exécuter une méthode.



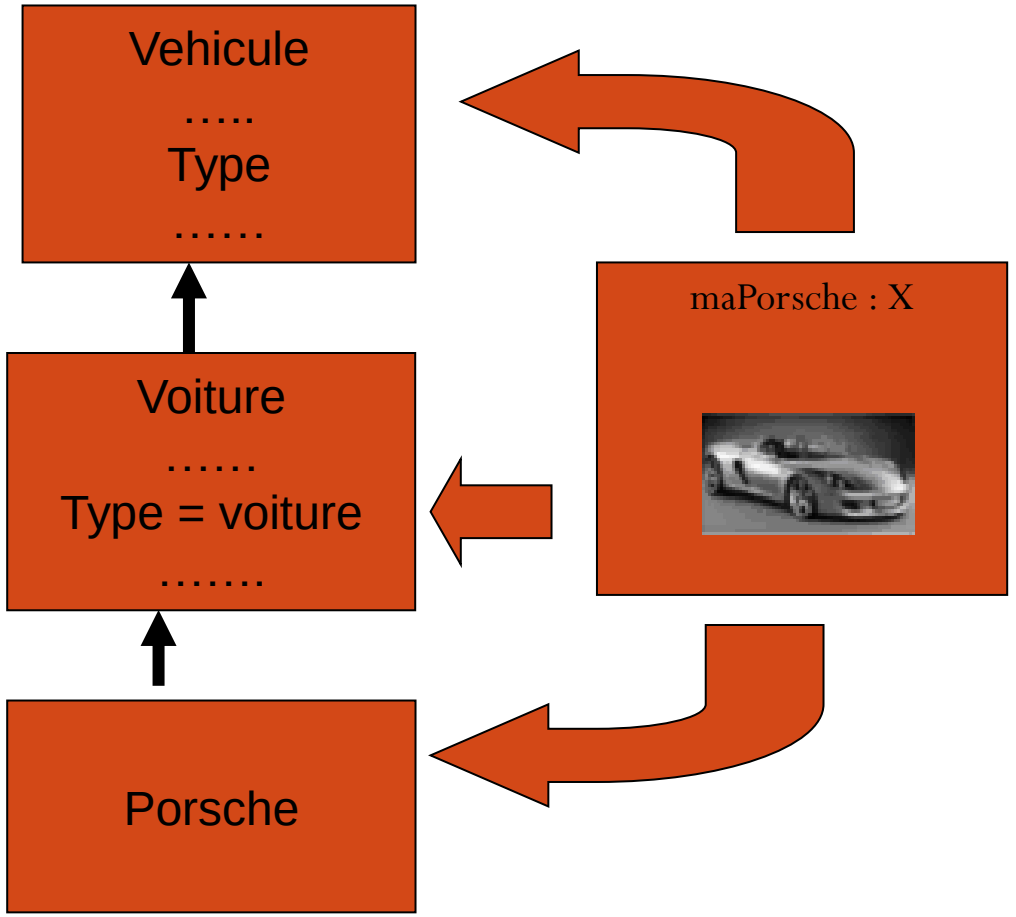
Héritage

- Une sous-classe est une Classe spécialisée qui hérite les attributs et méthodes d'une classe parente.



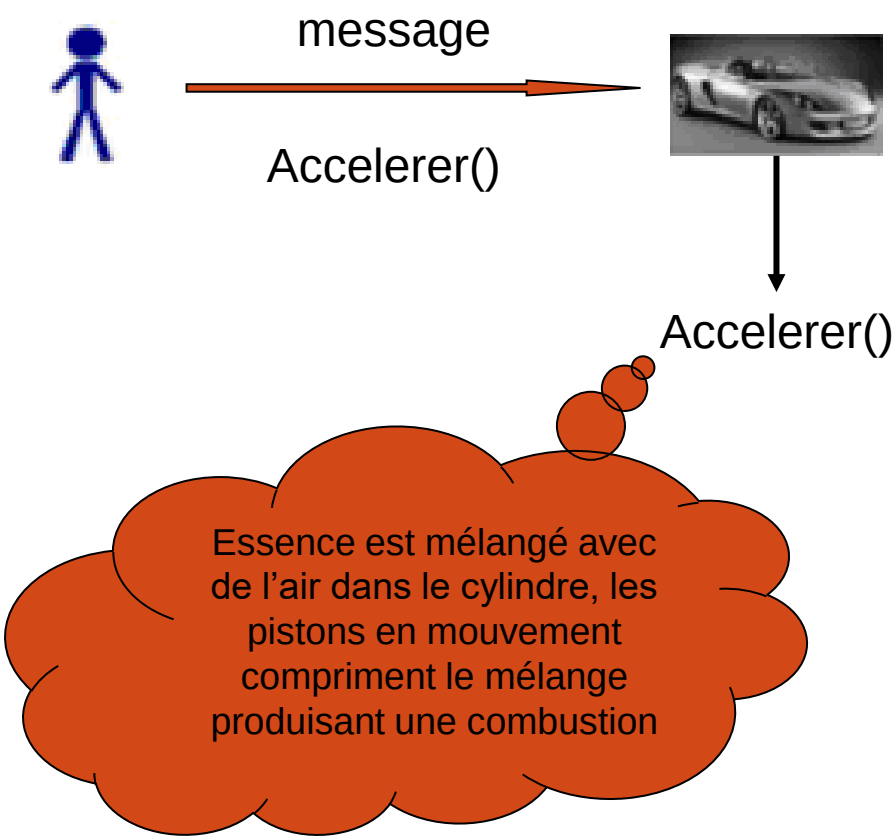
Abstraction

- L'abstraction est la pratique consistant à réduire des détails afin que l'on puisse se concentrer sur un Concept à la fois



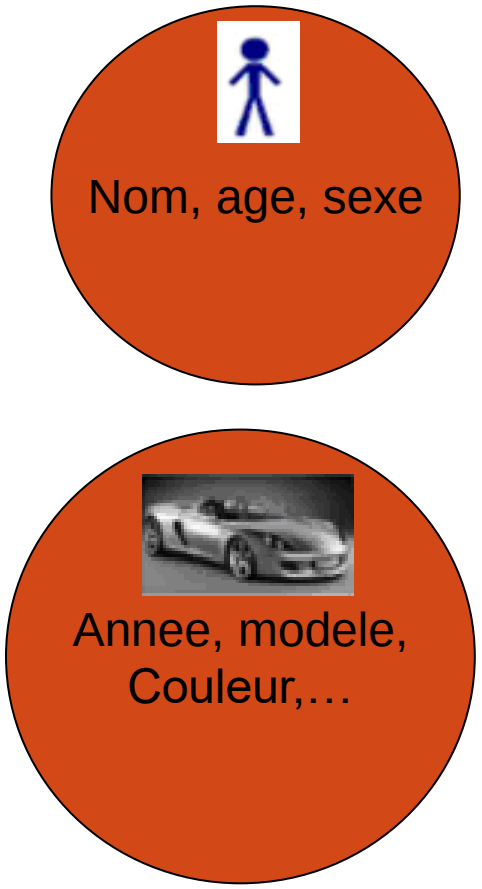
Encapsulation

- L'encapsulation cache les détails fonctionnels d'une classe aux objets qui lui envoie des messages



Encapsulation

- Protège l'intégrité d'un objet en empêchant l'accès direct à ses données
- Réduit la complexité d'un logiciel et accroit la robustesse en limitant les dépendances entre les composants du logiciel



Avantages de la POO

- les données sont protégés et accessibles d'une manière bien définie
- la modification du code des méthodes d'une classe n'influence en aucun cas les programmes externes utilisant cette classe
- les programmeurs peuvent réutiliser des composants existants

Bénéfices de la POO

- Robustesse : composants spécialisés construits ont plus de chance d'avoir été conçu correctement. Ils ont pu être teste intensivement.
- Gain de temps : réutilisation des composants.
- Moins de maintenance : la maintenance des composants réutilisés est à la charge de son développeur, donc moins de code à maintenir dans le programme

POO ou Programmation Procédurale?

- La programmation procédurale est bonne pour les problèmes pour lesquels il existe une solution bien définie (ex : calculer un factoriel d'un entier).
- POO pour les problèmes où il n'existe pas une seule bonne solution
- Ex 1: simulation du trafic routier : on analyse les comportements des objets (voiture, bus, train, vélo, feux)

  On obtient pas un résultat mais une modélisation du trafic, c.à.d. un modèle routier

  Le résultat est la plupart du temps différent d'une simulation à une autre

- Ex 2 : Windows : Pleins de fenêtres, avec comportements différents. Ce sont des objets. dans une fenêtre il existe d autres objets : boutons, etc.

POO ou Programmation Procédurale?

- En programmation procédurale, à chaque fois qu'on appelle une fonction avec le même paramètre, on obtient toujours la même réponse => pas de mémoire.
- Les objets se rappellent de ce qui s'est passé et agissent en conséquence